Laboratorio d'Informatica

Corso di Laurea Magistrale in Ingegneria per l'Ambiente e il Territorio

A.A. 2021/2022

Docente: Lorenzo Putzu

Lezione 07

Il linguaggio Python

Sets

- Un set o insieme è una collezione non ordinata di oggetti e può essere visto come un dizionario che contenga solo chiavi e non valori. Anche in questo caso gli elementi di un set possono essere valori numerici o Booleani, oppure nella maggior parte dei casi si tratta di stringhe.

- Lo stesso elemento **non** puo comparire più di una volta, per cui creare un insieme a partire da una lista o una sequenza equivale ad eliminarne i duplicati

- Anche gli insiemi come i dizionari sono sequenze **non ordinate**

Sets

- Un set puo essere creato come elenco di oggetti tra parentesi graffe e separati tra loro tramite la virgola, come ad esempio

```
frutta = {"mele", "pere", "banane", "susine"}
insieme = {"persone", 10, "numeri", False}
```

- Oppure può essere creato passando alla funzione `set` una sequenza ordinata (stringa o lista)

```
set(range(10))
set([10,"persona",False])
```

- Un set vuoto puo essere creato usando la funzione `set` senza argomenti (ricordiamoci che {} crea un dizionario vuoto).
- ATTENZIONE però all'uso della funzione `set` in quanto per definizione accetta un solo parametro di input e il suo comportamento cambia notevolmente in base al tipo di dato passato in input. Ad esempio le istruzioni che seguono hanno un esito completamente diverso
  - `set("ciao")`
  - `set(["ciao"])`

Sets: unione

- L'operazione di unione tra due insiemi restituisce un nuovo insieme composto dagli elementi presenti nel primo insieme più gli elementi presenti nel secondo insieme
  - **Sintassi:** set1 | set2
- In alternativa si può usare anche la seguente sintassi più mnemonica
  - **Sintassi:** set1.union(set2)
- Per esempio la sequenza di istruzioni

```
frutta = {"mele", "pere", "banane", "susine"}
fruttasecca = {"arachidi","noci","nocciole"}
frutta.union(fruttasecca)
```

- Darà vita ad un nuovo insieme che comprende gli elementi di entrambi insiemi (dato che non ci sono elementi uguali)
- Da notare che gli insiemi originali non vengono intaccati

Sets: intersezione

- L'operazione di intersezione tra due insiemi restituisce un nuovo insieme composto dagli elementi presenti in entrambi gli insiemi
  - **Sintassi:** set1 & set2
- In alternativa si può usare anche la seguente sintassi più mnemonica
  - **Sintassi:** set1.intersection(set2)
- Per esempio la sequenza di istruzioni

```
char = {"a", "p", "b", "s"}
char2 = {"a","n","b"}
char.intersection(char2)
```

- Darà vita ad un nuovo insieme che comprende gli elementi comuni ad entrambi gli insiemi
- Anche in questo casi gli insiemi originali non vengono intaccati

Sets: differenza

- L'operazione di differenza tra due insiemi restituisce un nuovo insieme composto dagli elementi presenti nel primo insieme ma che non sono presenti nel secondo insieme
  - **Sintassi:** set1 - set2
- In alternativa si può usare anche la seguente sintassi più mnemonica
  - **Sintassi:** set1.difference(set2)
- Per esempio la sequenza di istruzioni

```
char = {"a", "p", "b", "s"}
char2 = {"a","n","b"}
char.difference(char2)
```

- Darà vita ad un nuovo insieme che comprende gli elementi del primo insieme che non sono comuni anche al secondo
- Anche in questo casi gli insiemi originali non vengono intaccati

Sets:confronto

- Gli operatori == e != consentono di scrivere espressioni **condizionali** (il cui valore sarà True o False) consistenti nel confronto tra due set.
- **Sintassi**:
  - set1 == set2
  - set1 != set2
- dove set1 e set2 indicano due insiemi, che sono considerati identici se presentano gli stessi elementi

Sets: in e not in

- Anche per gli insiemi si possono utilizzare le espressioni **condizionali** che hanno lo scopo di verificare se un elemento sia presente o meno all'interno di un insieme
- **Sintassi**:
  - espressione in set
  - espressione not in set
- dove espressione indica una qualsiasi espressione Python e se il **valore** di espressione è presente tra gli elementi di set l'operatore `in` produce il valore True; in caso contrario produce False. Il comportamento dell'operatore `not in` è quello opposto.

Sets: issubset

- A volte però è necesario verificare che più di un elemento o espressione sia presente o meno all'interno di un insieme, o anche verificare se un certo insieme sia un sottinsieme di un altro insieme. Per questo scopo si usa il metodo *issubset* con la seguente sintassi
- **Sintassi**:
  - {espressione1,espressione2,...}.issubset(set)
  - set1.issubset(set)
- se il **valore** di tutte le espressioni o se il set1 è un sottoinsieme di set il metodo *issubset* produce il valore True; in caso contrario produce False.

Sets: iterazioni

- Anche per gli insiemi è possibile utilizzare le funzioni iterative così come le abbiamo viste per liste e diionari con la seguente sintassi

```
for v in s :
    sequenza di istruzioni
```

- v deve essere il nome di una variabile che non sia già usata per memorizzare dati all'interno dello stesso programma
- s deve essere un'espressione avente come valore un insieme
- sequenza di istruzioni è una sequenza di una o più istruzioni qualsiasi che di norma eseguono un'operazione sulla variabile v
- Tuttavia, nella maggior parte dei casi le istruzioni iterative sono semplicemente usate per scopi di stampa a video o su file, dato che anche in questo caso v contiene una **copia** degli elementi di s . Ma soprattutto dato che la maggior parte delle operazioni sugli insiemi possono essere fatte sull'insieme intero ma non su singoli elementi (non è indicizzabile)

Sets: funzioni predefinite

- Alcune metodi e funzioni predefinite di utilità generale sono le seguenti:
  - `len(set)`
    restituisce la lunghezza di un set
  - `set.add(elemento)`
    aggiunge elemento ad un set
  - `set.remove(elemento)`
    se elemento esiste lo rimuove dal set altrimenti da errore
  - `set.discard(elemento)`
    se elemento esiste lo rimuove dal set altrimenti il set rimane invariato
  - `set.clear()`
    rimuove ogni elemento dal set

Esercizi

1. Definire una funzione che riceva come argomento una lista e determini se in essa siano presenti valori duplicati (cioè valori che occorrono più di una volta), restituendo `True` oppure `False`

2. Modificare la funzione dell'esercizio della lezione precedente (Lez06_es03) in modo da evitare istruzioni iterative nidificate.

Tuple

- Una *tupla* è un tipo di dato molto simile alle liste in quanto permette di memorizzare una sequenza ordinata di valori che possono appartenente a tipi **qualsiasi**, anche diversi tra loro.
- A differenza delle liste le tuple (come le stringhe) sono **immutabili** perciò i loro valori non possono essere modificati.
- Una tupla si rappresenta nei programmi Python come una sequenza ordinata di valori scritti tra parentesi **tonde** e separati da virgole
- Esempi

```
(-5.3, 6, True)
(2*10, False, 'ciao')
()
```

Tuple

- Anche le tuple possono essere create per conversione utilizzando la funzione *tuple* a partire da un altro tipo di dato quindi ad esempio da una lista o da un intervallo creato con la classe range.
- Esempi

  tuple(range(10))

  tuple([10, True, ‘ciao’])

- Come le stringhe e le liste è un tipo **sequenza** per cui si possono utizzare tutte le operazioni che abbiamo visto in precedenza per questo tipo di dato.

Tuple: indicizzazione

- L'operatore di indicizzazione consente di accedere a ogni singolo elemento di una tupla, per mezzo dell'indice corrispondente secondo la **Sintassi**:

```
tupla[indice]
```

- Anche in questo caso è possibile utilizzare lo slicing per ottenere una porzione della tupla

- Ovviamente trattandosi di un tipo di dato immutabile non è possibile assegnare un nuovo valore al corrispondente indice o alla porzione selezionata della tupla.
  - Questo non vale per gli elementi nidificati all'interno della tupla: perciò nel caso una tupla contenga una lista gli elementi della lista potranno essere modificati

Tuple: concatenazione

- Questo operatore è analogo al corrispondente operatore del per stringhe e liste.
- **Sintassi:** tupla1 + tupla2
- La concatenazione produce una **nuova** tupla composta dagli elementi di tupla1 seguiti da quelli di tupla2, disposti nellostesso ordine in cui si trovano nelle due tuple. Ovviamente le tuple originali **non** vengono modificate.
- Ad esempio le seguenti istruzioni

```
tupla = (1,-5,4)
tupla + (1,2)
```

- Daranno vita ad una nuova tupla con i seguenti elementi (1, -5, 4, 1, 2)
- Inoltre è possibile creare una tupla che replicare gli elementi dela tupla originale con l'istruzione

```
tupla * 2
```

Tuple: funzioni predefinite

- Alcune funzioni predefinite di utilità generale sono le seguenti:
  - `len(tupla)`
    restituisce la lunghezza della tupla
  - `tupla.count(espr)`
    restituisce il numero di volte in cui espr appare nella tupla
  - `tupla.index(espr)`
    restituisce la posizione in cui espr appare per la prima volta nella tupla

Tuple: nidificate

- Anche le tuple come le liste possono contenere al loro interno delle altre tuple e in questo caso si parla di tuple **nidificate.** Possono essere create semplicemente dichiarando una tupla all'interno di un'altra tupla come già visto per le liste. Ad esempio con le istruzioni seguenti:

```
tupla = (5,10)
tupla1 = (1, True, (3, "ciao"))
tupla2 = (1, tupla)
```

- Vengono create due tuple (tupla1 e tupla2) che contengono una tupla nidificata. Un ulteriore modo per creare delle tuple nidificate è quello che viene chiamato impacchettamento in tupla, ad esempio con l'istruzione

```
tupla3 = tupla, 10
```

- creiamo una nuova tupla che contiene una tupla nidificata. La stessa operazione si può usare per impacchettare dati semplici in una tupla

```
tupla4 = 10, True
```

Sequenze spacchettamento

- Il procedimento inverso, chiamato spacchettamento di tupla si può utilizzare per estrarre dalle tuple sdei singoli valori. Perciò se abbiamo una tupla che definisce le coordinate di un punto possiamo estrarre i valori con una singola istruzione, ad esempio

```
x, y = tupla
```

- In generale questo tipo di operazione si può eseguire su tutti i tipi sequenza e prende il nome di spacchettamento di sequenza. L'unico vincolo è che la lista di variabili a sinistra abbia un numero di elementi pari alla lunghezza della sequenza.
- Si noti che l'assegnamento multiplo è in realtà solo una combinazione di impacchettamento in tupla e spacchettamento di sequenza!

Sequenze spacchettamento

- Grazie a questa procedura di impacchettamento e spacchettamento nel linguaggio Python è possibile creare delle funzioni che restituiscono più valori in modo molto semplice con la sintassi
  - `return(valore1, valore2,…)`
- Ovviamente la chiamata di funzione dovrà prevedere la memorizzazione dei valori restituiti in un numero uguale di variabili
  - `(var1, var2,…) = funzione(…)`
- altrimenti se il numero di variabili è superiore verrà restituiro un errore, mentre se in numero di variabili è inferiore l'operazione potrebbe andare in porto ma i valori rimarranno impacchettati nelle tupla

Funzioni di utilità generale

- La funzione *type* restituisce il tipo del dato che gli viene passato come argomento. Il suo utilizzo principale è in fase di debugging ma anche per fare alcuni controlli, come ad esempio:

```
type(espressione) is list
type(espressione) is not tuple
```

- La funzione *dir* invece permette di visualizzare l'elenco degli attributi e dei metodi disponibili per un determinato tipo di dato che gli viene passato come argomento, come ad esempio

```
dir([])    #restituisce l'elenco dei metodi e attributi di una lista
dir({})    #restituisce l'elenco dei metodi e attributi di un dizionario
```

- La funzione *help* fornisce delle informazioni più precise su uno specifico metodo come ad esempio

```
help([].insert)
help(().count)
```

Esercizi

3. Definire una funzione che, data una lista L, restituisca due nuove liste:
   - una ottenuta facendo "scorrere" di una posizione verso destra gli elementi di L e spostando l'ultimo elemento nella prima posizione. `[3,-2,4,20]` → `[20,3,-2,4]`
   - l'altra ottenuta facendo "scorrere" di una posizione verso sinistra gli elementi di L e spostando il primo elemento nell'ultima posizione. `[3,-2,4,20]` → `[-2,4,20,3]`
4. Definire una funzione che, data una lista di dizionari come il seguente, contenenti i dati sugli studenti che hanno sostenuto un certo esame: `{"matricola":"12345", "cognome":"Gui", "nome":"Ugo", "voto":27}`, restituisca il numero e la media aritmetica dei voti degli studenti che hanno superato l'esame, e la posizione nella lista degli studenti che hanno conseguito il voto 30 con lode (rappresentato dal valore 31).

Esercizi Ripasso

5. Definire una funzione che riceva come argomento una stringa contenente una frase (che si assume non contenere caratteri di punteggiatura) e stampi nella *shell* tutte le parole presenti in essa in ordine crescente della loro lunghezza
6. Scrivere una funzione che restituisca una lista contenente tutti i numeri primi compresi tra 2 e un dato intero n ricevuto come argomento, usando il metodo detto *crivello di Eratostene*: partendo dalla sequenza di tutti i valori tra 2 e n, si eliminano prima i multipli di 2, poi i multipli del valore successivo (tra quelli rimanenti), e così via
7. Scrivere una funzione che riceva come argomento una lista di numeri e restituisca il *secondo* valore maggiore presente in essa.
   Esempi: [3, 6, 1, 5] → 5, [4, -2, 7, 5, 7, 1] → 7